Enhancement of Cross Validation Using Hybrid Visual and Analytical Means with Shannon Function

Boris Kovalerchuk

Abstract The algorithm of *k*-fold cross validation is actively used to evaluate and compare machine learning algorithms. However, it has several important deficiencies documented in the literature along with its advantages. The advantages of quick computations are also a source of its major deficiency. It tests only a small fraction of all the possible splits of data, on training and testing data leaving untested many difficult for prediction splits. The associated difficulties include bias in estimated average error rate and its variance, the large variance of the estimated average error, and possible irrelevance of the estimated average error to the problem of the user. The goal of this paper is improving the cross validation approach using the combined visual and analytical means in a hybrid setting. The visual means include both the point-to-point mapping and a new point–to-graph mapping of the n-D data to 2-D data known as General Line Coordinates. The analytical means involve the adaptation of the Shannon function to obtain the worst case error estimate. The method is illustrated by classification tasks with simulated and real data.

Keywords *k*-fold cross validation · Machine learning · Visual analytics · Visualization · Multidimensional data · Shannon function · Worst case · Error estimate · Error rate · General line coordinates · Linear classifier · Hybrid algorithm · Interactive algorithm

1 Introduction

1.1 Preliminaries

Cross validation (CV) hold out estimate is a common way to evaluate the performance of classifiers in machine learning. In *k*-fold cross validation data are split into *k* equal-sized folds. Each fold is a validation/test set for evaluating classifiers learned

B. Kovalerchuk (✉)
Department of Computer Science, Central Washington University, Ellensburg, USA
e-mail: borisk@cwu.edu

O. Kosheleva et al. (eds.), *Beyond Traditional Probabilistic Data Processing Techniques: Interval, Fuzzy etc. Methods and Their Applications*, Studies in Computational Intelligence 835, https://doi.org/10.1007/978-3-030-31041-7_29

on the remaining $k-1$ folds. The error rate is computed as the average error across the k tests and is considered as an estimate of the error expectation. The empirical error on a test set in CV often is a *more reliable* estimate of the generalization error than the observed error on the training set [3]. The k-fold cross validation reduces the *computation* of a simple CV method known as leave-one-out cross validation [38]. Several variations of the k–Fold Cross Validation (KCV) for the Support Vector Machine (SVM) classification are compared experimentally in [1]. Parametric methods for comparing the performance of two classification algorithms evaluated by k-fold cross validation are proposed in [35] and strategy to find the global minimum CV error as a function of two SVM parameters in [10]. Selection of k for k-fold validation under some assumptions is explored in [2].

Four cross validation *schemes* are presented in [29], which are summarized below:

(1) Standard stratified cross validation (SCV) places an equal number of samples of each class on each partition to keep the same class distributions in all partitions.
(2) Distribution-balanced stratified cross validation (DB-SCV) keeps data distribution as *similar* as possible between the training and validation folds and maximizes the diversity on each fold to minimize the covariate shift.
(3) Distribution-optimally-balanced stratified cross-validation (DOB-SCV) is DB-SCV with the additional information used to choose in which fold to *place* each sample.
(4) Maximally-shifted stratified cross validation (MS-SCV) creates the folds that are as *different* as possible from each other. It tests the maximal influence partition-based covariate shift on the classifier performance by putting the *maximal shift* on each partition.

Here *covariate shift* means different distributions on the training and test sets [32], e.g., a unimodal distribution on the training set and a two-modal distribution on the testing/validation set.

This paper provides a justification for the use of the worst case estimates and Shannon Functions. The case studies show the examples of visual ways of worst case estimates in the data of different dimensions in combination with the analytical methods. This paper is organized as follows. Section 1 contains preliminaries, k-fold Cross validation challenges and process. Section 2 describes the method that includes the adaptation of the Shannon function (Sect. 2.1), discussion of alternative algorithms (Sect. 2.2), and the interactive hybrid algorithm (Sect. 2.3). Section 3 provides three case studies: on linear SVM and simplified Fisher Linear Discriminant Analysis (LDA) on modeled data in 2-D to illustrate the hybrid algorithm (Sect. 3.1); on LDA and visual classification in 4-D on Iris data (Sect. 3.2), and on GLC-AL and simplified LDA algorithms in 9-D on Wisconsin Breast Cancer Diagnostic data (Sect. 3.3). Section 4 contains discussion and conclusion.

1.2 *Challenges of* k-*Fold Cross Validation*

Challenges of cross validation have been analyzed for a long time. The representative publication is [7] where four sources of random variation in cross validation are identified.

- Selection of *validation data* to evaluate learning algorithms A and B. On a randomly selected validation data A can outperform B, though, on the whole, population A and B can be identical.
- Selection of *training data* to evaluate learning algorithms A and B. On a randomly selected training data A can outperform B, though on average A and B can be identical. Decision trees suffer from such instability even with adding or deleting few points.
- Internal randomness of the *algorithm*. Neural networks initiate random weights. The algorithm GLC-AL that we use in this paper randomly initiates coefficients.
- Randomly *mislabeled* a fraction of validation data. It is hard to expect that the algorithm will get fewer errors than this fraction.

Below we summarize more challenges specific for k-fold cross validation that are relevant to this paper.

Selecting *k*. The first question is *how to select k* for k-fold split. The larger k can lead to models with fewer errors on validation data due to larger training data. For instance, for $k = 10$, 90% of data are in training sets and only 10% are in the validation set in each training-validation pair. In contrast, smaller k can lead to models with more errors on validation data due to the smaller training data, e.g., for $k = 2$ we have 50%:50% split between training and validation data. The lower k can give a higher confidence in accuracy of the model on the validation data due to the larger number of cases in the validation set, but less confidence in accuracy of the model on the training data due to smaller training data. For $k = 2$ we have only two alternatives for a given split: (1) using the first half for training and second half for validation, or (2) vice versa.

Multiple *k*. Running k-fold cross validation for multiple k increases computation load, and still may not justify selection of a specific k when performance for different k varies significantly.

Selecting a split (partitioning). There are multiple ways to split data to k bins (folds). For instance, for $k = 2$ it is a number of combination $C(m, m/2) = m!/(m/2)!^2$, where m is the number of given n-D points. For a very small training set with $m = 100$ we have $C(100, 50) > 10^{29}$. This number of splits grows exponentially with m. The question is *how to select a particular split* out of these 10^{29} splits for $m = 100$. If we select only a single split out of these 10^{29} splits the accuracy of classification in this split may or may not be representative for the given dataset.

Multiple splits. Selecting multiple splits of data is computationally expensive with exponential grows with m. The question is *how many splits* to make and what k to keep. The use of the statistical criteria to evaluate the statistical significance of the accuracy of the result in a single split or a few splits can be questioned from

multiple viewpoints [7]. This is especially challenging for high-dimensional data. For instance, 100 points in 100-D space hardly represent the 100-D probability distribution function (pdf). Note that for images 100-D is just a way to represent a tiny image with 10×10 pixels in a gray scale.

Multiple criteria of accuracy. The question is *how to select a criterion* to estimate the error on both training and validation data. The estimate of the expected (average) error $E(e)$ used in k-fold cross validation may not be the best one for the user's task such as the tasks with high cost of individual errors. The alternatives are max error, min max error, weighed error and others.

In summary the main problems with k-fold are that:

(i) many splits that are difficult for prediction on the verification data will *not be tested* [7],
(ii) estimated average errors can be *biased* [7],
(iii) estimated variance of average errors can be *large* and/or *biased* [6, 7],
(iv) estimates of the average error and its variance in (ii) and (iii) can be *insufficient* or even *irrelevant* to the supervised learning problem that is of user's interest.

The first three problems are well documented in the literature on statistical machine learning. The theorem proved in [6] states that there *exists no* universal (valid under all distributions) unbiased estimator of the variance of k-fold cross validation. Multiple attempts have been made to address k-fold problems under different additional *assumptions*. The examples include a modification of k-fold known as 5x2CV cross validation to decrease the bias and improve t-statistics used for evaluation [7] and unbiased variance estimates under restrictive assumptions on the distribution of cross-validation residuals [9]. Other more recent studies are listed in Sect. 1.1. To the best of our knowledge much less was done for the problem (iv) in both probabilistic and deterministic settings.

The problem (iv) is considered in this paper with the use of the Shannon function. This problem is related to the Maximally-shifted stratified cross validation (MS-SCV) listed above as schema (4). It is found in extensive experiments on real data in [29] that: (1) MS-SCV produces a *much worse accuracy* than all other partitioning strategies, and (2) cross validation approaches that limit the partition-induced covariate shift (DOB-SCV, DB-SCV) are *more stable* when running a single experiment, and need a lower number of iterations to stabilize. These results illustrate well the problem. We can limit the covariate shift in cross validation to get a more stable result on validation data. However, nobody can guaranty us that on new unseen data the covariance shift will be limited or limited in the same way. It is simply out of our control in many real world tasks.

Therefore, the stable result under such limits on the partition-induced covariate can be biased showing a lower error rate than it can be on the real test data. If a user will get estimates of expected error rate in schemas that:

- limit the partition-induced covariate shift ("average" case) and
- do not limit them, but are looking for the bounds for "worst" and "best" cases,

then the risk of using a given learning algorithm with "average" case will be balanced by more complete information. This is the ultimate goal of this paper.

1.3 k-*Fold Cross Validation Process*

In this section we define the k-fold cross validation algorithm and its challenges. Let D be a set that consists of m_1 samples of class 1 and m_2 samples of class 2 with each sample is an n-D point and let k be a number of folds (bins) used to split data. Table 1 illustrated k-fold cross validation algorithm for $k = 10$ and 1000 samples (500 from class 1 and 500 from class 2). Assume that the first 500 n-D records in D are samples of class 1 already randomly ordered. Assume that the second 500 records in D are samples of class 2 that are also randomly ordered. For $k = 10$ in each of 10 pairs (training data, validation data) 90% of data are in the training dataset and 10% are in the validation dataset.

Below we describe **steps of *k*-fold algorithm** in general terms with comments on alternatives that it does not explore when run for a given k:

(1) Select the number of bins equal to k. Commonly k is between 2 and 10. The given k is only one of these alternatives,
(2) Select a way to split D into k bins with about m_1/k and m_2/k samples of classes 1 and 2, respectively in each bin with the total of about m/k points in each bin. The term "about" is used here to reflect the fact that m_1/k and m_2/k may not be integers and need to be adjusted to be integers. The k-fold algorithm uses the random split as the way to split D. It produces *one split* of D to k bins out of the many possible splits that can be produced randomly or non-randomly.

Table 1 Example of k-fold cross validation algorithm for $k = 10$

Step 1	Assign $k = 10$
Step 2	Form k bins (folds) Bin 1: Bin 11: samples 1–50 from class 1, Bin 12: samples 1–50 from class 2 Bin 2: Bin 21: samples 51–100 from class 1 Bin 22: samples 51–100 from class 2 ….. Bin k: Bin $k1$ samples 451–500 from class 1, Bin $k2$: samples 451–500 from class 2
Step 3	Form training validation pair P(i) = <Tr(i), Val(i)> For every i: 1, 2, … k, Val(i) = Bin(i) and all other bins are in Tr(i) For instance, in <Tr(1). Val(1)> Val(1) = Bin1 and all other bins are in Tr(1)
Step 4	For every i compute the error $e(i)$ on Val(i) obtained by the algorithm Alg_j trained on Tr(i) Compute average of all $e(i)$ as an estimate of the expectation E of e, $E(e)$ and estimate its statistical significance relative to another algorithm or random prediction

(3) Form k pairs (training data, validation data) with about $(k-1)$ *m/k training* samples and about *m/k validation* samples in each pair by using a split of D to k-bins from (2). Validation data for different pairs do not overlap. Other splits on (3) would generate other pairs. Moreover, even for the given split, these k pairs are a part of a much larger set of training, validation pairs with j *m/k* samples for training and $(k-j)$ *m/k* samples for validation. For instance, for $k = 10$ and $j = 3$ the pair contains 70% of D in the training data and 30% of D in the validation set. The reason for using only k pairs in the k-fold ensures that the test data do not overlap. 10-fold does not test more challenging pairs 70%:30%, but only 90%:10% split pairs.

(4) Select the function to estimate prediction error and compute this function using all k pairs from (3). The standard k-fold selects the estimate of expected (average) error $E(e)$ as described in Table 1. The alternatives are max error, min max error and others.

2 Method

2.1 Shannon Function

Below we formalize a way to evaluate the worst case as a compliment to k-fold estimates of the average error. It is done by adaptation of the minimax **Shannon function** [30] originally proposed for analysis of the complexity of switching circuits as Boolean functions. The Shannon function measures the *complexity of the most difficult function*. It was used in the evaluation of complexity of computation of Boolean functions by analog circuits [33]. The complexity $L(f)$ of a function f is the lower bound of the complexity of circuits realizing f. The function $L(n)$, equal to the maximum complexity of functions of n arguments is called a Shannon function [13]. In particular, this function was applied to find an algorithm A_j that restores the worst (most complex) monotone Boolean function of n-variables for the smallest number of queries [11, 16].

Consider a labeled dataset D and a set of machine learning algorithm $\{A_j\}_{j\in J}$. Let $\{D_i\}_{i\in I}$, $1 = \{1, 2, \ldots, m\}$ be a set of splits of D to <Training data, Validation data> **pairs**. k-fold cross validation split is one of them. Each D_i is a pair of training and validation data, $D_i = (Tr_i, Val_i)$. $A_{jv}(D_i)$ is the **error rate** on validation data Val_i produced by A_j when A_j is trained on the training data Tr_i from D_i. The adaptation of the **Shannon function** $S(I, J)$ to supervised learning problem is defined as follows

$$S(I, J) = \min_{j\in J} \max_{i\in I} A_{jv}(D_i) \tag{1}$$

The algorithm A_b is called ***S*-best algorithm** if

$$S(I, J) = \min_{i \in I} A_{bv}(D_i) \tag{2}$$

In other words, the *S*-best algorithm produces fewer errors on validation data on its worst *k*-fold splits among $\{D_i\}$ than other algorithms on their worst *k*-fold splits among $\{D_i\}$.

Let $\boldsymbol{D}_a = \{D_i{:}i \in I_a\}$ be a set of *all* passible *k*-fold splits for given *k* and data *D*, i.e., $k - 1$ folds (bins) with the training data and one fold (bin) with the validation data. In contrast with the standard *k*-fold validation, here the validation sets for different D_i can *overlap*. Let $\boldsymbol{D}_T = \{D_i{:}i \in T\}$ is some set of splits.

Statement. If $\boldsymbol{D}_T = \{D_i{:}i \in T\} \subseteq \boldsymbol{D}_a$ then $S(I_a, J) \leq S(I_T, J)$

This statement follows directly from definitions of these terms. For instance, if $S(I_T, J) = 0.2$ then adding more splits can give us a better split D_r in D_a such that $A_{jv}(D_r) < 0.2$ for some A_j.

In other words, for each $\boldsymbol{D}_T$ the value of $S(I_a, J)$ provides a **low bound** for $S(I_T, J)$. Similarly, for $\boldsymbol{D}_a$ the value of $S(I_T, J)$ provides an **upper bound** for $S(I_a, J)$. A standard *k*-fold split $\boldsymbol{D}_K = \{D_i{:}i \in K\}$ is one of $\boldsymbol{D}_T$. How close the bounds are to the actual worst case depends on the specific $\boldsymbol{D}_T$ and D_a. At least the average error rate for $\boldsymbol{D}_K$ can be computed quickly enough. Computing error rates for multiple $\boldsymbol{D}_K$ produced by random or non-random splits of data into folds will give several bounds.

Asymptotically this will lead to the actual Shannon worst case,

$$D_w = \arg(\min_{j \in J} \max_{i \in I} A_{jv}(D_i)) \tag{3}$$

Split D_w is called the *worst case split for S-best algorithm* A_b.

$$D_w = \arg(\max_{i \in I} A_{bv}(D_i) \tag{4}$$

Informally, the worst case split is a split, which is most difficult for the *S*-best algorithm which produces fewer errors on validation data than other algorithms on their worst splits from $\{D_i\}$.

Split D_b is called the *best case split for S-best algorithm* A_b

$$D_h = \arg(\min_{i \in I} A_{jv}(D_i) \tag{5}$$

Informally, the best case split is a split, which is easiest for the *S*-best algorithm which produces fewer errors on the validation data than the other algorithms on their worst splits from $\{D_i\}$.

Split D_m is called the *median split for S-best algorithm* A_b,

$$D_h = \arg(\operatorname{median}_{i \in I}\big(A_{jv}(D_i)\big)) \tag{6}$$

Informally, the median split for the *S*-best algorithm produces the error rate that is close to the average error rate among $\{D_i\}$ for A_b algorithm.

The worst and best estimates (4) and (5) compliment (6) and the traditional *k*-fold expectation estimate evaluated by the *t*-test statistics. This is especially useful when the expectation has a high variance. Both the worst case and best case estimates provide the "bottom line" of the expected errors. As we mentioned above, for the tasks with a high cost of individual error, it is very important.

2.2 Alternative Algorithms

This section analyzes options for algorithms to find worst, best and median splits defined above. The options are:

- *brute force* algorithms,
- specialized automatic algorithms that exploit known *structural information* about data,
- interactive algorithms that exploit 2-D *visual representation* of n-D data, and
- *hybrid* algorithms that combine automatic and interactive visual algorithms.

The brute force algorithms require exploration of the number of alternatives, which grow exponentially with the size of *D*. Therefore, such algorithms are of practical interest only for very small datasets. Specialized algorithms must be developed for each type of structural information about data. Thus, the approach based on the structural information is labor intensive and not scalable. Interactive and hybrid algorithms are most promising and will be explored in this paper. We focus on the hybrid algorithms as this allows combining the advantages of *automatic* and *interactive visual* algorithms.

There are two major types of 2-D visualizations of n-D data available in the hybrid approach:

(1) each n-D *point* is mapped to a 2-D *point* (we denote this mapping as P-P), and
(2) each n-D *point* is mapped to a 2-D structure such as a *graph* (we denote this mapping as P-G).

Principal Component Analysis (PCA) [12, 36], Multidimensional Scaling (MDS) [25], Self-Organized maps (SOM) [14], RadVis [31] are examples of (1), and Parallel Coordinates (PC) [15], and General Line Coordinates (GLC) [18–20] are examples of (2). The P-P representations (1) are not reversible (lossy), i.e., in general there is no way to restore the n-D point from its 2-D representation. In contrast PC and GLC graphs are reversible [19].

The next issue is preserving n-D *distance* in 2-D. While such P-P representations as MDS and SOM are specifically designed to meet this goal, in fact, they only minimize the mean difference in distance between the points in n-D and their representations in 2-D. PCA minimizes the mean-square difference between the original points and the projected ones [36]. For individual points the difference can be quite large. For a 4-D hypercube SOM and MDS have Kruskal's stress values $S_{som} = 0.327$ and S_{mds}

= 0.312, respectively, i.e., on average the distances in 2-D differ from distances in n-D over 30% [8].

Such high distortion of n-D distances (loss of the actual distance information) can lead to misclassification, when such corrupted 2-D distances are used for the classification in 2-D. This problem is well known and several attempts have been made to address by controlling and decreasing it, e.g., for SOM in [36]. In medical and engineering diagnostic tasks, as well as defense object classification tasks with high cost of error, it can lead to disasters and loss of life.

In contrast, the distance between graphs in 2-D can be defined to *preserve the distance* between all n-D points not only minimize the average difference of distances. Below we explain it.

Let A* and B* be graphs for n-D points A = (a_1, a_2, …, a_n) and B = (b_1, b_2, …, b_n). In PC, each a_i and b_i of A and B is represented as a node of the graph. If the distance between a_i and b_i is e, $|a_i - b_i| = e$ then the distance between nodes a^*_i and b^*_i in PC is the same, $|a^*_i - b^*_i| = e$ due to design of PC. Thus, $D(A^*, B^*)$ is defined as

$$D(A^*, B^*) = ||A - B|| = (\sum_{i=1}^{n}(a_i - b_i)^2)^{1/2}$$

The same is true for other General Line Coordinates that map each a_i to a graph's node a^*_i. For those GLC that map each pair (a_i, a_{i+1}) to a graph's node $a^*_{i,i+1}$ and each pair (b_i, b_{i+1}) to a graph's node $b^*_{i,i+1}$ the distances between these nodes is a standard Euclidian distance in 2-D, $D(a^*_{i,i+1}, b^*_{i,i+1}) = ((a_i - b_i)^2 + (a_{i+1} - b_{i+1})^2)^{1/2}$. The squared distance between graphs $D^2(A^*, B^*)$ is defined as the sum of all squared $D^2(a^*_{i,i+1}, b^*_{i,i+1})$. Thus, $D(A^*, B^*)$ is as before, $D(A, B) = ||A - B||$, just it is computed using pairs,

$$D(A^*, B^*) = ||A - B|| = \left(\sum_{i=1}^{n/2}(D(a^*_{i,i+1}, b^*_{i,i+1})^2\right)^{1/2}$$

Note that if n is odd, the last coordinate x_n is repeated to get the even n. The formula above assumes such even n. Informally if n-D points A and B are close to each other, then the graphs A^* and B^* in GLC are also close to each other. In P-P representations it is not guaranteed. For this reason, the visual means that we use are based on the *General Line Coordinates*.

In the hybrid approach below the visualization guides both:

(1) getting the information about the *structure* of data, and
(2) finding the *worst, best* and *median* split of data into the training–validation pairs.

In current machine learning practice, 2-D representation is commonly used for illustration and explanation of the ideas of the algorithms such as SVM or LDA, but much less for actual discovery of n-D rules due to the difficulties to adequately represent the n-D data in 2-D, which we discussed above.

2.3 Interactive Hybrid Algorithm

Below we propose non-random heuristic ways to generate spits to get better estimates of worst, best, and median case error estimates. While there are always counterexamples for heuristic ideas these ideas are more successful for finding worst, best, and median cases than random splits used in typical cross validation. The first step is setting up a threshold for samples from opposing classes to be considered as closely located.

Worst case heuristic is to include closely located points of opposing classes to *validation data* Val, but not to training data Tr. The intuition behind it is that closely located points from opposing classes have higher chance to be misclassified if not included to training data.

Best case heuristic is to include closely located points of opposing classes to *training data* Tr, but not to validation data Val. The intuition behind it is that difficult for classification closely located points from opposing classes have higher chance to be classified correctly if used for training the classifier.

Median case heuristic. Worst and best splits described above are mixed proportionally with the splits that do not have Tr and Val from worst and best splits. Alternatively, none from best and worst splits are included.

Other heuristics to decrease computations are building best and worst cases using only points that are located on the frames of the convex hulls of opposing classes and only inside of convex hulls for the average cases.

If classes are separable (convex hulls do not overlap), and the distance between closest points of convex hulls is large (say, comparable with the length of the convex hulls), then it is likely that the worst, best, and median cases will produce error-free discriminant functions for multiple classification algorithms. In this situation, all these algorithms will be *S*-best algorithms, and the cross validation exploration can be minimized.

In contrast, when the classes are closely located or overlap, extensive cross validation is required. This includes a situation when a search for the worst case splits led to a split with a large error rate on validation data. If the further exploration produced only a single much better split, then it must be justified beyond its high accuracy before using it for prediction. Such justifications can be establishing that the discovered model is explainable, which adds the confidence.

The steps of **first part** of the **interactive hybrid algorithm for the *S*-best algorithm,** which is discovering the data *structure* are as follows:

(S1) Visualize n-D data in 2-D.
(S2) Select border points of each class, color them in different colors.
(S3) Outline classes by constructing envelopes in the form of a convex or a non-convex hull.
(S4) Outline (a) overlap areas L for overlapped classes or (b) select closest areas C for separable classes.
(S5) Compute the size of the overlap areas L or areas C of the closest samples.

(S6) Set up ratio of training-validation data, |Tr||/|Val|, e.g. 90%:10% with (|Tr|| + |Val|)/|Val| = k.

The steps of **second part** of the interactive hybrid algorithm for the **worst case** (**IH-W**) **of S-best algorithm** are:

(W1) Form Val as areas L or C.
(W2) Adjust (increase or decrease) L or C to make |L| = |Val|, or |C| = |Val|.
(W3) Form training data Tr = D\Val and pair <Tr, Val>.
(W4) Apply each Algorithm A_j to Tr to construct discrimination function F.
(W5) Apply F to Val to get error rate A_{jv}(Val).
(W6) Record A_{jv}(Val) and find max(A_{jv}(Val)), $j \in J$.
(W7) Repeat (W1)–(W6) to get values {max A_{jv}(Val_i)} $i \in I$ for a set of training-validation pairs {D_i}.
(W8) Find the Shannon worst case split, $\min_{i\in I} \max_{j\in J}$ (A_{jv}(Val_i)) and algorithm A_b that provides this split.

The interactive algorithm for the **best case** (**IH-B**) **of S-best algorithm** is:

Use algorithm A_b from step W8 to get $\min_{i\in I}$ (A_{bv}(Val_i)).

The interactive algorithm for the **median case** (**IH-A**) **of S-best algorithm** is:

Use algorithm A_b from step W8 to get $\text{median}_{i\in I}$ (A_{bv}(Val_i))

Note: norm |X| can be computed as the actual number of cases from D in the area X or as size of the area X depending on the density of the points of calluses in the areas.

3 Case Studies

The problem in n-D space is that we do not see n-D data, and need visual tools to represent n-D data in 2-D space. Figures below, in case studies, show how the visual means support finding worst and best cases of splits with the use of the interactive hybrid algorithm presented in this section.

3.1 Case Study 1: Linear SVM and LDA in 2-D on Modeled Data

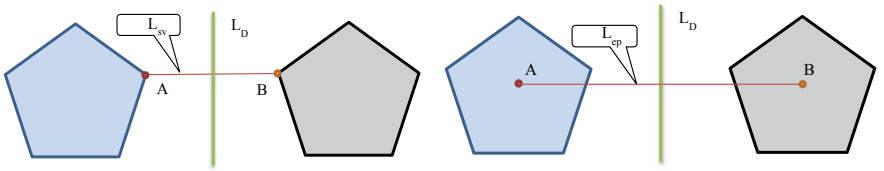
In this section, we assume a point-to-point (P-P) representation of n-D data in 2-D such as PCA, MDA, and SOM. The interactive hybrid algorithm is demonstrated for the search of the worst case estimates in cross validation for the two classification algorithms. These two algorithms are the linear SVM and the simplified Fisher Linear Discriminant Analysis (LDA). First we illustrate both algorithms with the examples

in Figs. 1 and 2. For the linear SVM we use its geometric interpretation [4, 5], which is based on the closest support vectors of the two classes.

In Fig. 1a **linear SVM** uses line $L_{sv}(A, B)$ that connects two closest *support vectors* (SV) A and B from opposing classes (blue and grey pentagons that constitute data *D*). The line $L_{sv}(A, B)$ is used to build a discrimination line L_D. Line L_D bisects line $L_{sv}(A, B)$ in the middle and is orthogonal to $L_{sv}(A, B)$.

In Fig. 1b **simplified Fisher LDA** uses the *average points* for each class (points A and B), connects them with line L_{ap}. Then the orthogonal line L_D bisects line L_{ap} in the middle. The line L_D serves as the discrimination line.

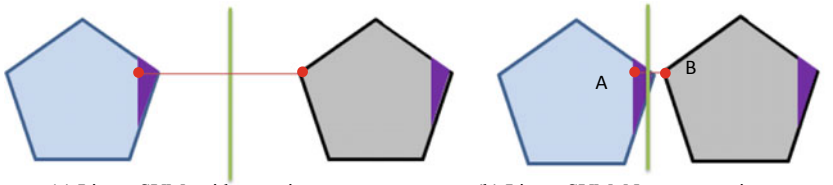
In Fig. 1, both algorithms produce the same green discrimination line, which is error free before any cross validation splits of these data. Figure 2 shows the results of linear SVM and simplified LDA for one of 10-fold splits D_i of the data *D* in the cases of wide and narrow margins between the classes (pentagons). In both pictures, the two violet triangles form a test set (total 10% of both pentagons). The remaining



(a) The red line connects the closest support vectors and green line bisects it in the middle to serve as a SVM linear classifier

(b) The red line connects centers and green line bisects it in the middle to serve as a LDA linear classifier.

Fig. 1 Two separable classes with wide margin classified by linear SVM and simplified LDA. All points of each class are in the respective convex hulls (blue and grey pentagons)



(a) Linear SVM: wide margin case

(b) Linear SVM: Narrow margin case

(c) Simplified LDA wide margin case

(d) Simplified LDA: narrow margin case

Fig. 2 Two separable classes with wide and narrow margin classified by linear SVM and simplified LDA. In SVM the red line connects closest support vectors from opposing classes. In LDA it connects centers of training data of classes. The green lines that bisect these lines in the middle serve as a SVM and LDA linear classifiers, respectively. In the case of the narrow margin both classifiers are not error free

parts of the pentagons are the training data of this D_i pair. Figure 2a, c shows that in the case of the wide margins both the algorithms SVM and LDA are also error-free, while producing different discrimination, functions. Thus, Fig. 2a, c provides examples of the **best case** of the split for these algorithms.

In contrast, Fig. 2b, d show the errors in the case of a narrow margin with a larger error for the linear SVM than for the LDA. We cannon state that these figures show the **worst case** for any of these algorithms, but the smaller error for LDA can serve a **bound** for *S*-best algorithm among these two algorithms. Other splits can have larger errors.

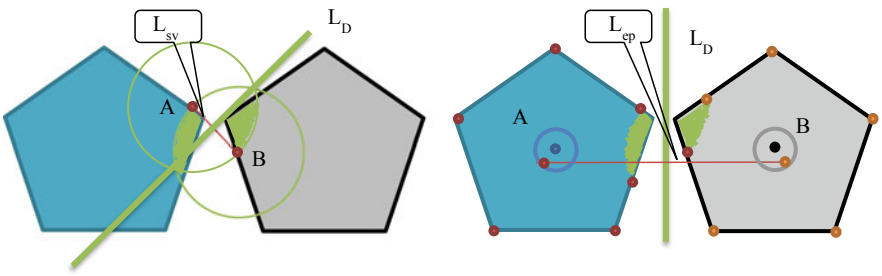
Now we will show how these examples are related to the first part of the interactive algorithm—visual discovery of the data structure. The results of steps S1–S5 are shown in all parts of Fig. 2, Steps S3 are shown in Fig. 2b in addition to steps S1–S5.

Worst case. For the worst case, the steps W1–W5 are also illustrated in these examples. In Sect. 2.3 we outlined a heuristic for finding a worst case split, which is finding closely located points of opposing classes to be included to validation data Val. The violet areas in Fig. 2 satisfy this heuristic. They were selected by visual analysis of classes in Fig. 1.

Figure 3 illustrates the next W steps of the interactive algorithm, where the adjustment is starting from W7 to adjust/modify the validation data. Figure 3 shows the result of the modification, which generates more errors than the split in Fig. 2, providing a **stronger bound** for the worst error. The general idea of designing such stronger estimates is modifying visually the current split.

In Fig. 3 the green areas form the validation data (5% of the blue pentagon and 5% of the grey pentagon). The points A and B on the edge of the green areas belong to the training data, but all other points of those inner edges belong to the validation data. These edges are segments of the perimeters of the circles of equal radiuses with centers in A and B. Thus points A and B are the closest support vectors of the training data from opposing classes.

The green linear SVM discrimination line, produced using these A and B, has significant error in both training and validation data, because A and B are located asymmetrically (A higher than B). In Fig. 2b, A and B are at the same height. It is



(a) Linear SVM (b) Simplified LDA

Fig. 3 Modification of validation area from Fig. 2b

visible from the picture that larger difference in height leads to more errors in these data.

The analytical part of the step W2 at this stage uses a binary search with substeps: (W21) finding the middle point on the pentagon edge where the point A is located. Substep W22 is finding radius R such that the circle with R cuts a green area in the grey pentagon equal to the 5% of that pentagon area. It is done by several iterations. Substep W23 is getting a candidate for the point B in the crossing of the grey pentagon and the circle. Substep W24 is drawing a circle of radius R from B, and computing the green area in the grey pentagon. Substep W25 is checking if this area is greater than the 5% of the pentagon area, and moving point A to the right on its edge to the middle of that half of that edge, otherwise A is moved to the middle of the left half of the edge. Now substeps W22–W25 are repeated with binary splits of the edge until the difference from 5% will be small enough to stop.

How to ensure that this process will converge? Step 2 will find the required area for every location of point A. If B gives more than 5%, point A is moved to the right. If new B still gives more than 5%, point A is moved further to the right. If finally B gives less than 5%, point A is moved back until 5% is reached within the required accuracy. For the case when B gives less than 5%, the sequence is similar.

Statement. Figure 3a is the worst case for linear SVM, when the two closest SV of the two full pentagons shown in Fig. 1 are removed from the training data and placed into the validation data.

Proof. Any split of the pentagons in Fig. 3 into the training and validation data that keeps the closest SV in the training set produces the same discrimination line (the green line in Fig. 1). This line is the optimal one because it provides error-free discrimination of the pentagons. Thus, to get a line with errors we need to remove at least one of the points A and B from the training data. Figure 3a shows such a case when both original A and B from Fig. 1a are removed from the training data and the new closest support vectors A and B are identified.

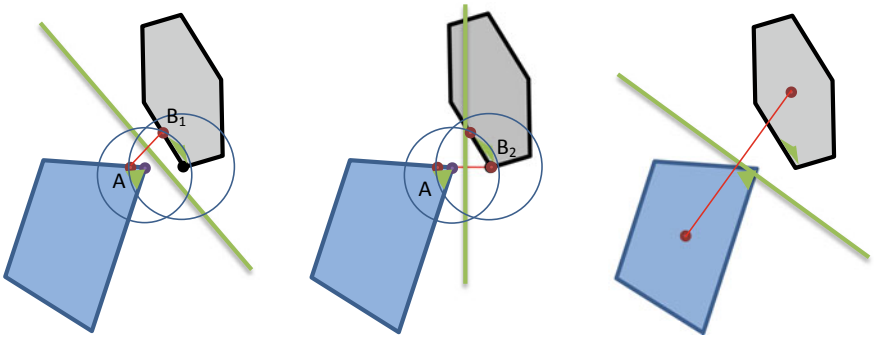
Let for a given point B a classifier with more errors than in Fig. 3a exists; it must have its own SV in class 1 that is closest to B. Denote it as C. With this C no training data from the blue pentagon can be in the green area other than point C, because these points are closer to point B than C. Otherwise, C is not the closest SV to B. This green area without C must be outside of the training data and must belong to the validation data. In the 10-fold design for pentagons, the validation data must be no greater than 5% of the pentagon area. Point A is selected at exactly 5% of the blue pentagon area. Thus, point C cannot differ from A. Therefore, Fig. 3a is the worst case when both original SV A and B are removed from the training data.

Figure 3b shows the result of the simplified LDA for the same validation data. This result is the best case for the simplified LDA because this pair D_i is error-free (see green discrimination line in Fig. 3b). For this D_i we have $A_{v\mathrm{SVM}}(D_i) > A_{v\mathrm{LDA}}(D_i) = 0$. Thus, LDA is the winner as the S-best algorithm for this D_i. For the previous D_i in Fig. 2 we also have $A_{v\mathrm{SVM}}(D_i) > A_{v\mathrm{LDA}}(D_i)$, but $A_{v\mathrm{LDA}}(D_i) > 0$. Denote D_i from Fig. 2 as D_2 and from Fig. 3 and D_3. In this notation, $A_{v\mathrm{SVM}}(D_2) > A_{v\mathrm{LDA}}(D_3)$. Therefore, LDA is the winner as the S-best algorithm for both D_2 and D_3 in comparison with linear SVM.

Generalization for arbitrary convex hulls. The example above with two pentagons at specific locations shows that knowing the *specific structural information* about the data *it is possible to derive the exact worst case split* for the given k, and for simplified LDA and linear SVM. This approach can be generalized to any convex hull not only equal pentagons at the specific locations. Figure 4 illustrates this for two arbitrary convex hulls. It uses the same way of designing the worst case validation data (selecting closest areas of two classes) for linear SVM as in Fig. 3. In Fig. 4 there are two closest SV B_1 and B_2 in the grey hexagon to point A, which is in the blue rectangle. The SVM discriminant line for B_1 is error-free, but the discriminant line for LDA in Fig. 4c is not. Linear SVM is a winner for D_i in Fig. 4 that we denote as D_4, $A_{v\text{LDA}}(D_4) > A_{v\text{SVM}}(D_4)$. In this example, we build the discriminant lines only for two closest SV from two classes. We do not consider the case when a single discriminant line is constructed for several closest SVs while it can be done similarly.

Discussion. What is important in the examples in this case study? It is not the existence of the k-fold cross validation where one algorithm is better than another. It is a fact that it was fund visually. The probability of this discovery is very low under the blind random assigning of data to bins in the k-fold algorithm. The following numerical example shows this.

Assume that we have 1000 samples of the two classes in the two pentagons in Fig. 3. Thus, each bin (fold) in 10-fold will contain 100 samples. Also assume that in each (Tr, Val) pair D_i, training data contain 900 samples and the validation data contain remaining 100 samples (50 samples from class 1 and 50 cases from class 2). Each pentagon has only 5 nodes. We assume that all of them are among 500 samples in the dataset D. With the random selection the probability to get the training or validation set with one specific node from blue pentagon (denoted as node A) is equal to 1/500. Respectively the probability p to get training or validation data with two specific nodes A and B is low $(1/500)^2$.



(a) Linear SVM best case. (b) Linear SVM worsened case (c) Simplified LDA case

Fig. 4 Linear SVM and simplified LDA with different error rates for arbitrary convex hulls. Green areas are the verification data

Moreover, in Fig. 2b points A and B not only belong to training data, but are closest support vectors from two opposing classes. The probability of this is even lower under the random process of putting samples to the bins. It means that getting a case D_i that we have in Fig. 3 is unlikely by a random process. This case is one of the worst cases for linear SVM in these data. In general, it means that if we are not able to discover such D_i then the k-fold will not allow us to see the difference between these algorithms.

Next, even if such D_i is included, the difference between average error estimates for two algorithms will likely be statistically insignificant if both algorithm equally accurate on the remaining nine training-validation pairs. This is a motivation for using the Shannon function and for search of the worst cases or at least *estimates the bounds of the worst cases*.

Why is it important to search for such rare worst training-validation pairs D_i? The ultimate goal of machine learning is *generalization* beyond the given data D to unseen data. The existence of worst training–validation pairs with large error indicates that the algorithm A_j *does not capture a generalization pattern* in some situations on given data D.

This increases the chances of misclassification on unseen data too. In the tasks with the *high cost of an individual error* (e.g., in medicine and defense) such situations must be traced and analyzed before use in real applications. For instance, if the S-best algorithm defined in terms of the Shannon function on a set of selected splits $\{D_i\}$ is not error-free then the areas where those errors occurred can be treated differently than the error-free areas; It can be:

(1) refusal to classify data from those areas,
(2) use other machine learning algorithms,
(3) adding more data and retraining on extended data,
(4) cleaning existing data,
(5) modifying features,
(6) use other appropriate means including manual classification by experts.

The case study in this section uses 2-D modeled 2-D data of a given structure. As was pointed out at the beginning of this section, to make it useful for real n-D machine learning tasks 2-D data can be obtained from real n–D data by PCA, MDS, SOM and other point-to-point matching visualization algorithms.

3.2 Case Study 2: LDA and Visual Classification in 4-D on Iris Data

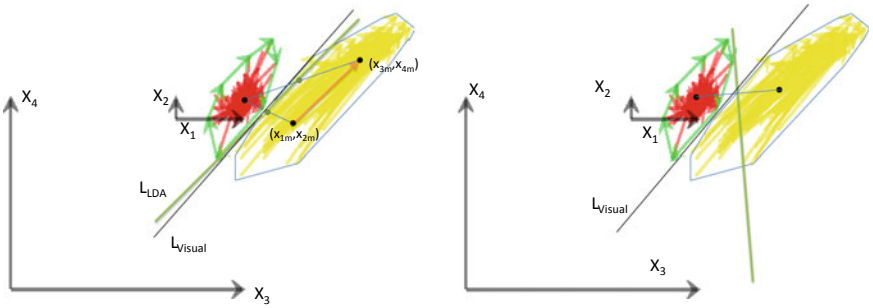
The case study in this section is based on the graph representation of n-D samples in 2-D, not on a single 2-D point representation of n-D points considered in case study 1. This graph representation is called Parametrized Shifted Paired Coordinates (PSPC) [19, 20]. In PSPC a 4-D data points (x_1, x_2, x_3, x_4) is represented in 2-D as arrows where the beginning of the arrow is point (x_1, x_2) in coordinates (X_1, X_2) and

the end in the point (x_3, x_4) in coordinates (X_3, X_4). In Fig. 5, both coordinate systems are shown. The example of such an arrow in an orange arrow defined by pairs (x_{1m}, x_{2m}) and (x_{3m}, x_{4m}) in Fig. 5a. This point is the mean of all 4-D points of class 2 that we will call the center of class 2. The mean of all 4-D points of class 1 is another arrow. However, the location of coordinates pairs (X_1, X_2) and (X_3, X_4) on 2-D plain is selected in PSPC in such a way that this arrow is collapsed to a single point. This single point is shown as a black dot in the middle of the red blob that represents class 1. This parametrization of the location is described in [19]. For an n-D point with $n > 4$, the arrow will be transformed into a sequenced of arrows (directed graph). For an n-D point that is used as an anchor for the parametrization, this graph collapses into a single 2-D point in the same way as described above for the 4-D.

In Fig. 5, Iris 4–D data [24, 26] of two classes are shown as arrows in PSPC anchored in the center of class 1. Then two linear classification algorithms are applied. Figure 5a shows the simplified LDA classifier (green line) and a visually constructed classier (thin black line) for the 4-D data. For comparison, Fig. 5b shows the result of applying the simplified LDA (dark green line) if data in those convex hulls would be 2-D data. The visually constructed discriminant line is the same thin black line as in Fig. 5a. In this case black dots in Fig. 5b are 2-D centers of classes 1 and 2. The simplified LDA algorithm in Fig. 5b is the same as the one used in case study 1. It produced a larger error than the visually constructed discriminant (thin black line), which is error-free.

In contrast, for the 4-D case in Fig. 5a, the centers are arrows, not points, and the middle of them is not a single point, but the line that connects the two green points. The linear classifier (green line) is the extension of that line with a very small error.

What is important in the example in Fig. 5a? It is the abilities to build a visual classifier (black line), to build visually a simplified LDA (green line), and be able to compare the level of error visually without extensive computation. It also allows the chopping visually of parts of the blobs to set up them as new validation data and build new visual discrimination lines, compare the errors, and find worst and best



(a) Simplified LDA discriminant for 4-D data in PSPC (dark green line) and visually constructed discriminant (thin black line). The orange arrow is the middle 4-D point of class 2.

(b) Simplified LDA discriminant (dark green line) for 2-D data and visually constructed discriminant (thin black line). Black dots are 2-D centers of classes 1 and 2.

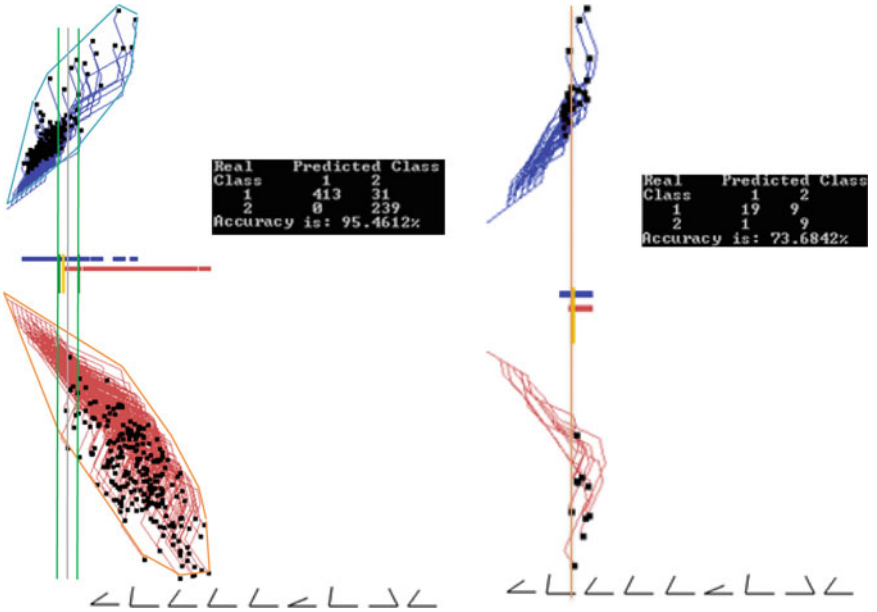
Fig. 5 Iris data in parametrized shifted paired coordinates (PSPC) anchored in class 1

splits similarly to shown in case study 1. Purely computationally, it would require massive combinatorial computations, and we still may not find the worst cases.

3.3 *Case Study 3: GLC-AL and LDA in 9-D on Wisconsin Breast Cancer Diagnostic Data*

The case study in this section is also based on the graph representation of the n-D samples in 2-D, not on a single 2-D point representation of an n-D point. For this study, Wisconsin Breast Cancer Diagnostic (WBC) dataset was used [24, 26] with 9 attributes for each record and the class label which was used for classification. The samples without missing values include 444 benign cases and 239 malignant samples. Figure 6 shows the samples of screenshots, where these data are interactively visualized, and classified with a linear classifier using GLC-L algorithm [20], and GLC-AL algorithms [22] with the accuracy over 95% on these data. The malignant cases are drawn in red and benign in blue. For convenience of reading these algorithms are presented in the appendix.

Below we show a way to get a worst case for the linear GLC-AL algorithm. The comparison with the other algorithms is conducted by developing and using the 2-D



(a) Two thresholds are set for selecting the overlapping cases.

(b) Overlapping cases from the interval between the two thresholds from (a).

Fig. 6 9-D Wisconsin breast cancer data in lossless GLC-L visualization and classification by algorithm GLC-AL by an algorithm

versions of the linear SVM and LDA in the GLC-L visualization. Here we w use the convex hulls in 2-D not in n-D. We also use the interactive GLC-IL algorithm [22], where the training process includes adjusting a threshold without finding the new coefficients. In Fig. 6, the number of samples in the overlap area from both classes is small (5.6%), and therefore can be visually analyzed quickly.

In Fig. 6a, the GLC-AL linear classifier misclassified 31 samples with all of them from class 1 when all data (444 benign cases and 239 malignant samples) were used for training. The selected overlap area contains 38 samples (4.5%, with 28 samples from class 1, and 10 samples from class 2).

According to step W1 of the algorithm IH-W, we form the validation set Val as a set of samples in the overlap area L. We keep Val equal to L without adjustment, skipping the step W2. Next we use a shortcut for steps W2–W5, which allows us to get a bound for the error rate $A_{jv}(L)$, where A_{jv} is the GLC-L algorithm applied to $Val = L$ trained on Tr. The result of this shortcut is presented in Fig. 6b. It shows the overlapping cases L, selected in Fig. 6a and the accuracy of classification of samples from L, when all of them and only them are used as training data. At the first glance, running GLC-L on L as training data, not validation data, contradicts steps W2-W5, which require to running L as validation data. The trick is that, *training* GLC-L on L as training data, we expect to get a smaller error rate on L than *running the linear model* on L, constructed by GLC-L on training data Tr without any data from L in Tr.

In Fig. 6b, the accuracy is 73.68% (error rate 0.2632) with L as training data. The error rate 0.2632 is the upper bound for the error rate $A_{jv}(L)$, $A_{jv}(L) \leq 0.2632$. We cannot get a bound with the larger number of errors than 0.2632 for the algorithm GLC-L, if we continue to run GLC-L on the overlap area L for more epochs. It follows from the design of GLC-L. GLC-L keeps coefficients with the current lowest error rate. Having the error rate equal to 0.2632 GLC-AL will update it only by finding a smaller error rate, not a larger one.

This conclusion was made under assumption that we use L as Tr. Now we need to explore what will happen with the other splits when L is only a part of Tr, not equal to Tr. Can we get another error rate r for GLC-L, say $r = 0.3$, which is greater than 0.2632 for these other splits and respectively another upper bound for $A_{jv}(L)$? If such greater r exists our previous claim, that we cannot get more errors with GLC-L, will be wrong.

We cannot get such greater r for the same reason as above. The design of GLC-L will not allow it. We already have a linear model in Fig. 6a that classified all samples from Tr = D\L with zero error rate, where D is the total given dataset. Thus GLC-AL algorithm trained on Tr data that include L will only keep linear models that classify L better because for samples outside L GLC-AL already obtained models with zero error rate.

This shortcut can be applied for any GLC-L data. If such upper bound is a tolerable error rate, then we can apply the coefficients found by GLC-AL on TR\L as training data for classification of new data. Thus, steps W2–W5 of the algorithm IH-W for GLC-L can be simplified.

To compare the bound for GLC-L with the bounds for linear SVM and LDA steps, W4–W6 must be run for these algorithms. The algorithm with the smallest bound will be a candidate for the *S*-best algorithm on these data. In addition to this analytical option, an interactive option exists for the modified and simplified versions of linear SVM and LDA algorithms that work with 2-D GLC-L visual representations of n-D data. Both algorithms follow the steps used in case study 1 with two differences: (1) convex hull constructed by GLC-L algorithm are used, and (2) the overlap area is defined by the location of the last node of the graph (marked by black squares). This way to identify the overlap area was used in Fig. 6.

Linear SVM in GLC-L visualization uses closest support vectors (SV) from two classes in GLC-L. For overlapping convex hulls of two classes we use the overlap area that is identified by a user interactively using two thresholds (see green lines in Fig. 6a). Two closest nodes of graphs from two different classes in the overlap area are called closest support vectors. If the overlap area is empty (the case of linearly separable classes) then two closest nodes of the frames of two convex hulls are called closest support vectors. Having two closest support vector A and B we build a line that connects them and a line that bisects than in the middle and orthogonal to the first line. The closest nodes are defined in the projection line of the last point to the horizontal line (see yellow line in Fig. 6b).

For the LDA we compute A as an average point in the projection on the point of class 1 to the horizontal line and point B the same for the class 2. Then the middle point C between A and B is used to construct the discrimination line. It is shown in Fig. 6a as a grey line.

What is important in the example in Fig. 6 is the same as in case studies 1 and 2—the abilities to build a visual classifiers (in this case for 9-D), and be able to compare error rates visually. It also allows chopping visually overlapping parts by setting up thresholds interactively and using these folds to construct validation data for the worst case.

4 Discussion and Conclusion

While cross validation is very useful, it needs to be improved to deal with its deficiencies such as leaving untested many potentially difficult-for-accurate-prediction splits. It is challenging due to a need to keep its advantage of faster computation. This paper had shown a hybrid way to improve cross validation by using combined visual and analytical means. We use both the well-known point-to-point and new point-to-graph mapping of n-D data to 2-D data. The main *benefit* of this hybrid approach is leveraging the abilities of the human visual system to *guide the discovery of* patterns in 2-D. This includes discovering splits of n-D data in 2-D visualization of these data. This approach creates an opportunity to avoid a blind computational search of worst splits among the exponential number of alternatives that can be the case in the pure computational analytical approach. In essence, the visual approach brings *additional information* about the n-D data structure that the pure computational approach lacks.

Adding such information from the visual channel can be viewed as a way to add more features and relations to the data, sometimes called privileged information [34], or prior domain knowledge [27, 28]. The difference is that both privileged information and domain knowledge typically are assumed to not be present in the original data. In contrast, the visual channel makes the hidden information already present in n-D data be readily available via the interactive process.

While this visual opportunity exists, it requires the relatively simple visualization for humans to be able to discover a pattern in them, i.e., within the abilities of the human visual channel. The ways to simplify the visual patterns in the General Line Coordinates are proposed in [20]. Such ways should be applied before in concert with the interactive search for worst case splits in cross validation.

The focus on worst case splits and adaptation of the Shannon function bring a new formal validation task that covers both validation with or without cross splits depending on a set of split used. Three cases studies illustrate the proposed approach for different dimensions.

The main justification for the use of worst case estimates and Shannon Functions is three-fold:

(1) Existence of the tasks with a *high cost* of individual errors (e.g., medicine and defense);
(2) Existence of the tasks with a relatively low cost of individual error and a low average error rate, but the high error rate for the worst case splits;
(3) Abilities to limit the application of the algorithm in the worst folds avoiding the risky predictions.

In (1) and (2) the use of the average error rate can be too optimistic and risky where the worst case estimate serves as warning, while (3) allows preventing risky decisions. We may have two algorithms A and B with the average error rates with a statistically insignificant difference, but A has much smaller worst case error rate than B. This can be a reason to prefer A for the classification of new samples, because A was able to discover better difficult patterns than B showing stronger generalization ability. In addition while error rate for A is better than for B in the worst case, in some worst folds it can be too big. The prediction in these folds can be blocked for both A and B.

In Sect. 1 we listed the several challenges for k-fold cross validation. These challenges are related to: (1) selecting the number of folds k and running multiple k, (2) selecting data split, running multiple splits and missing multiple splits that left untested, (3) large variance of error rates, (4) bias in estimated average errors and its variance, and (5) insufficiency or *irrelevance of* estimated average errors (multiplicity of criteria of accuracy).

The proposed hybrid approach allows dealing with these challenges as follows. First $k = 2$ is used to provide an upper bound of the worst error rate for all the other k for the given algorithm A. Then we increase k until the worst case bound will be below threshold T_{worst} selected by a user for the given task. This k and k above it are considered acceptable. On the other extreme, with $k = m$ (leave-one-out split), where m is the number of samples, we consider another threshold T_{best}, and decrease

k until the best error rate will be still below T_{best}. Assume that we find k that satisfies both the T_{worst} and T_{best}. Such k ensures that we have fewer errors in both the worst and best cases, than the allowed thresholds for them. For instance, we can find that for k = 8 the worst error rate is bounded by 0.18 and the best error rate is bounded by the error rate 0.05, with average error rate as 0.12 with its variance ±0.02. In other words, we have a wider interval [0.05, 0.18] than the average interval [0.10, 0.14].

The computational support of visual exploration and visual support of analytical computations are important parts in this hybrid approach to avoid brute force search. It is important that in the examples in the case studies, the bounds for the worst splits were found by visual exploration without blind brute force computational search, despite rarity of these splits. This includes a quick visual judgment that the error rate in one split is greater than in another one. A user can find visually a large overlap area of two classes and chop it to form several validation folds, e.g., getting 10-fold cross validation splits. This confirms our main statement that brute force search is not mandatory and is avoidable using an appropriate visualization.

The future studies are toward making hybrid interactions more efficient and natural in the computational and visual aspects, but not limited by them going to more general data science approaches [21]. This includes adding speech recognitions to interactions allowing a user to give oral commands such as "decrease slightly the overlap area", "shift the overlap area to the right", "make an about 5% area on the top of the convex hull" and so on. This will require formalization of the linguistic variables involved in these commands in the spirit of the Computing with Words (CWW) approach [17, 37]. More complex commands such as "decrease *slightly* the overlap area, and shift the overlap area to be *close* to the envelope frame" will require more sophisticated uncertainty aggregation techniques [23] from probability theory, fuzzy logic and interval analysis.

Appendix

For convenience of reading this article, the appendix below presents the GLC-L algorithm from [20] and GLC-AL algorithm from [22], which are used in Sect. 3.3.

Appendix 1: Base GLC-L Algorithm

Let $K = (k_1, k_2, \ldots, k_{n+1})$, $k_i = c_i/c_{\max}$, where $c_{\max} = |\max_{i=1:n+1}(c_i)|$, and $G(\mathbf{x}) = k_1x_1 + k_2x_2 + \cdots + k_nx_n + k_{n+1}$. Here all k_i are normalized to be in $[-1, 1]$ interval. The following property is true for F and G: $F(\mathbf{x}) < T$ if and only if $G(\mathbf{x}) < T/c_{\max}$. Thus F and G are equivalent linear classification functions. Below we present the steps of the base visualization *algorithm* called *GLC-L* for a given linear function $F(\mathbf{x})$ with the given coefficients $C = (c_1, c_2, \ldots, c_{n+1})$.

Step 1: *Normalize* $C = (c_1, c_2, \ldots, c_{n+1})$ by creating as set of normalized parameters $K = (k_1, k_2, \ldots, k_{n+1})$: $k_i = c_i/c_{\max}$. The resulting normalized equation $y_n = k_1x_1 + k_2x_2 + \cdots + k_nx_n + k_{n+1}$ with the normalized rule: if $y_n < T/c_{\max}$, then **x** belongs to class 1 else **x** belongs to class 2, where y_n is a normalized value, $y_n = F(\mathbf{x})/c_{\max}$. Note that for the classification task, we can assume $c_{n+1} = 0$ with the same task generality. For regression we also deal with all the data normalized, e.g., if actual y_{act} is known, then it is normalized too, $y_{act}/c_{\max}$ for comparing with y_n.

Step 2: *Compute all angles* $Q_i = \arccos(|k_i|)$ of absolute values of k_i and locate coordinates $X_1 - X_n$ in accordance with these angles as shown in Fig. 7 relative to the horizontal lines. If $k_i < 0$ then coordinate X_i is oriented to the left, otherwise X_i is oriented to the right (see Fig. 7). For a given n-D point $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ draw its values as *vectors* $\mathbf{x}_1, \mathbf{x}_2, \ldots, \boldsymbol{x}_n$ in respective coordinates X_1–X_n (see Fig. 7).

Step 3. *Draw vectors* $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n$ *one after another*, as shown on the left side of Fig. 7. Then *project* the last point for $\mathbf{x}_n$ onto the horizontal axis U (see a red dotted line in Fig. 7). To simplify visualization axis U can be collocated with the horizontal lines that define the angles Q_i as shown in Fig. 8.

Step 4.

Step 4a. For regression and linear optimization tasks repeat step 3 for all n-D points as shown in the upper part of Fig. 8.

Step 4b. For the two-class classification task, repeat step 3 for all the n-D points of classes 1 and 2 drawn in different colors. Move points of class 2 by mirroring them to the bottom with axis U doubled as shown in Fig. 8. For more than two classes, Fig. 1 is created for each class, and m parallel axes U_j are generated next to each other similar to Fig. 8. Each axis U_j corresponds to a given class j, where m is the number of classes.

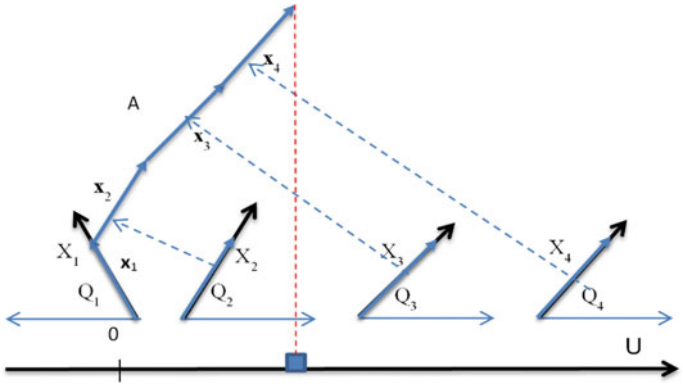


Fig. 7 4-D point A = (1, 1, 1.2, 1.2) in GLC-L coordinates X_1–X_4 with angles (Q_1, Q_2, Q_3, Q_4) and vectors $\mathbf{x}_i$ shifted to be connected one after another, and the end of last vector projected to the black line. X_1 is directed to the left due to negative $\underline{k}_1$. Always, the coordinates for negative k_i are directed to the left

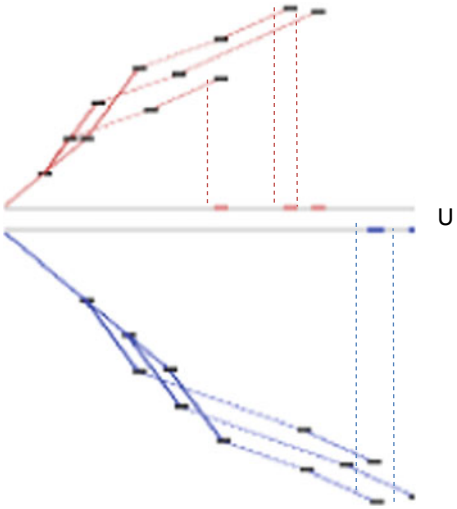


Fig. 8 Result with axis X_1 starting at axis U and repeated for the second class below it

Step 4c. For the multi-class classification task, conduct step 4b for all n-D points of each pair of classes i and j drawn in different colors, or draw each class against all other classes together.

This algorithm uses the property that $cos(arccos\ k) = k$ for $k \in [-1, 1]$, i.e., projection of vectors $\mathbf{x}_i$ to axis U will be $k_i x_i$ and with consecutive location of vectors $\mathbf{x}_i$, the projection from the end of the last vector $\mathbf{x}_n$ gives a sum $k_1x_1 + k_2x_2 + \cdots + k_nx_n$ on axis U. It does not include k_{n+1}. To add k_{n+1}, it is sufficient to shift the start point of $\mathbf{x}_1$ on axis U (in Fig. 7) by k_{n+1}. Alternatively, for the visual classification task k_{n+1} can be omitted by subtracting k_{n+1} from the threshold.

Appendix 2: Algorithm GLC-AL for Automatic Discovery of Relation Combined with Interactions

The GLC-AL algorithm differs from the Fisher Linear Discrimination Analysis (FDA), Linear SVM, and Logistic Regression algorithms in the criterion used for optimization. The GLC-AL algorithm directly maximizes accuracy,

$$A = (\mathrm{TP} + \mathrm{TN})/(\mathrm{TP} + \mathrm{TN} + \mathrm{FP0} + \mathrm{FN}),$$

which is equivalent to the optimization criterion used in the linear perceptron] and Neural Networks in general. In contrast, the Logistic Regression minimizes the Log-likelihood. Fisher Linear Discrimination Analysis maximizes the ratio of between-class to within-class scatter. The Linear SVM algorithm searches for a

hyperplane with a large margin of classification, using the regularization and the quadratic programming.

For the practical, GLC-AL uses a simple random search algorithm that starts from a randomly generated set of coefficients k_i, computes the accuracy A for this set, then generates another set of coefficients k_i again randomly, computes A for this set, and repeats this process m times. This is Step 1 of the algorithm shown below. A user runs the process m times more if it is not satisfactory.

Step 1:

```
Step 1:
    best_coefficients = []
    while n > 0
      coefficients <- random(−1, 1)
      all_lines = 0
      for i data_samples:
            line = 0
            for x data_dimensions:
                  if coefficients[x] < 0:
                        line = line – data_dimensions[x]*cos(acos(coefficients[x]))
                  else:
                        line = line + data_dimensions[x]*cos(acos(coefficients[x]))
            all_lines.append(line)
            //update best_coefficients
            n--
```

Step 2: Projects the end points for the set of coefficients that correspond to the highest A value (in the same way as in Figure 4) and prints off the confusion matrix, i.e., for the best separation of the two classes.

Step 3:
Step 3a:
1: User moves around the class separation line.
2: A new confusion matrix is calculated.
Step 3b:
1: User picks the two thresholds to project a subset of the dataset.
2: n-D points of this subset (between the two thresholds) are projected.
3: A new confusion matrix is calculated.
4: User visually discovers patterns from the projection.
Step 4: User can repeat Step 3a or Step 3b to further zoom in on a subset of the projection or go back to Step 1.

Validation process. In the current implementation, GLC-AL uses 10 *different* 70–30% *splits*, with 70% for the training set, and 30% for the validation set in each split. Thus GLC-L has the same 10 tests of accuracy as in the typical 10-fold cross validation, but 70–30% splits are more challenging than the tasks with 90–10% splits in 10-fold cross validation.

These 70–30% splits are selected by using the permutation of data. The *splitting process* is as follows:

(1) indexing all m given samples from 1 to m, $w = (1, 2, \ldots, m)$,
(2) randomly permuting these indexes, and getting a new order of indexes, $\pi(w)$.
(3) picking up the first 70% of indexes from $\pi(w)$,
(4) assigning the samples with these indexes to be the training data,
(5) assigning the remaining 30% of samples to be validation data.

This splitting process also can be used for a 90–10% split, or other splits.

References

1. D. Anguita, A. Ghio, S. Ridella, D. Sterpi, K-fold cross validation for error rate estimate in support vector machines, in *DMIN* (2009 Jan), pp. 291–297
2. S. Arlot, M. Lerasle, Choice of V for V-fold cross-validation in least-squares density estimation. J. Mach. Learn. Res. **17**(208), 1–50 (2016)
3. A. Blum, A. Kalai, J. Langford, Beating the hold-out: bounds for k-fold and progressive cross validation, in *Proceedings of the Twelfth Annual Conference on Computational Learning Theory* (ACM, 1999 Jul 6), pp. 203–208
4. K.P. Bennett, C. Campbell, Support vector machines: hype or hallelujah? ACM SIGKDD Explorations Newsl **2**(2), 1–13 (2000)
5. K.P. Bennett, E.J. Bredensteiner, Duality and geometry in SVM classifiers, in *ICML* (2000 Jun 29), pp. 57–64
6. Y. Bengio, Y. Grandvalet, No unbiased estimator of the variance of k-fold cross-validation. J. Mach. Learn. Res. **5**, 1089–1105 (2004)
7. T.G. Dietterich, Approximate statistical tests for comparing supervised classification learning algorithms. Neural Comput. **10**(7), 1895–1923 (1998)
8. W. Duch, R. Adamczak, K. Grąbczewski, K. Grudziński, N. Jankowski, A. Naud, *Extraction of Knowledge from Data Using Computational Intelligence Methods* (Copernicus University, Toruń, Poland, 2000). https://www.fizyka.umk.pl/~duch/ref/kdd-tut/Antoine/mds.htm
9. Y. Grandvalet, Y. Bengio, *Hypothesis Testing for Cross-Validation* (Montreal Universite de Montreal, Operationnelle DdIeR, 2006 Aug 29), p. 1285
10. B. Gu, V.S. Sheng, K.Y. Tay, W. Romano, S. Li, Cross validation through two-dimensional solution surface for cost-sensitive SVM. IEEE Trans. Pattern Anal. Mach. Intell. **39**(6), 1103–1121 (2017)
11. G. Hansel, Sur le nombre des functions Booléenes monotones de n variables. C.R. Acad. Sci., Paris, **262**(20), 1088–1090 (1966)
12. J. Jolliffe, *Principal of Component Analysis* (Springer, 1986)
13. N.A. Karpova, Some properties of Shannon functions. Math. Notes Acad. Sci. USSR **8**(5), 843–849 (1970)
14. T. Kohonen, *Self-Organizing Maps* (Springer, Berlin, Germany, 1995)
15. A. Inselberg, *Parallel Coordinates: Visual Multidimensional Geometry and Its Applications* (Springer, 2009)
16. B. Kovalerchuk, E. Triantaphyllou, A. Despande, E. Vityaev, Interactive learning of monotone Boolean functions. Inf. Sci. **94**(1–4), 87–118 (1996)
17. B. Kovalerchuk, Quest for rigorous combining probabilistic and fuzzy logic approaches for computing with words, in *On Fuzziness. A Homage to Lotfi A. Zadeh*, vol. 1 (Studies in Fuzziness and Soft Computing Vol. 216), edited by R. Seising, E. Trillas, C. Moraga, S. Termini (Springer, Berlin, New York, 2013), pp. 333–344
18. B. Kovalerchuk, Visualization of multidimensional data with collocated paired coordinates and general line coordinates, in *Proceedings of SPIE 2014*, vol. 9017. https://doi.org/10.1117/12.2042427
19. B. Kovalerchuk, Visual cognitive algorithms for high-dimensional data and super-intelligence challenges. J.: Cogn. Syst. Res. **45**, 95–108 (2017)
20. B. Kovalerchuk, V. Grishin, Adjustable general line coordinates for visual knowledge discovery in n-D data. Inf. Vis. (2017). https://doi.org/10.1177/1473871617715860
21. B. Kovalerchuk, M. Kovalerchuk, Toward virtual data scientist, in *Proceedings of the 2017 International Joint Conference on Neural Networks* (Anchorage, AK, USA, 14–19 May 2017), pp. 3073–3080

22. B. Kovalerchuk, D. Dovhalets, Constructing interactive visual classification, clustering and dimension reduction models for n-D data. Informatics **4**(23) (2017). http://www.mdpi.com/2227–9709/4/3/23
23. V. Kreinovich (ed.), *Uncertainty Modeling, Studies in Computational Intelligence*, vol. 683 (Springer, 2017)
24. M. Lichman, *UCI Machine Learning Repository* (University of California, School of Information and Computer Science, Irvine, CA, 2013). https://archive.ics.uci.edu/ml. Accessed on 15 June 2017
25. J. Kruskal, M. Wish, *Multidimensional Scaling* (Sage Publications, 1978)
26. M. Lichman, *UCI Machine Learning Repository* (University of California, School of Information and Computer Science, Irvine, CA, 2013). Parkinson's, https://archive.ics.uci.edu/ml/datasets/. Accessed on 15 June 2017
27. T. Mitchell, Introduction to machine learning, in *Machine Learning* (McGraw-Hill, Columbus, 1997)
28. T.J. Mitchell, S.Y. Chen, R.D. Macredie, Hypermedia learning and prior knowledge: domain expertise vs. system expertise. J. Comput. Assist. Learn. **21**(1), 53–64 (2005)
29. J.G. Moreno-Torres, J.A. Sáez, F. Herrera, Study on the impact of partition-induced dataset shift on *k*-fold cross validation. IEEE Trans. Neural Netw. Learn. Syst. **23**(8), 1304–1312 (2012)
30. C.E. Shannon, The synthesis of two-terminal switching circuits. Bell Syst. Tech. J. **28**, 59–98 (1949)
31. J. Sharko, G. Grinstein, K. Marx, Vectorized Radviz and its application to multiple cluster datasets. IEEE Trans. Vis. Comput. Graph. **14**(6), 1427–1444 (2008)
32. H. Shimodaira, Improving predictive inference under covariate shift by weighting the log-likelihood function. J. Stat. Plan. Inference **90**(2), 227–244 (2000)
33. G. Turán, F. Vatan, On the computation of Boolean functions by analog circuits of bounded fan-in, in *Proceedings of the 35th Annual Symposium on Foundations of Computer Science, 1994* (IEEE, 1994 Nov 20), pp. 553–564
34. V. Vapnik, A. Vashist, A new learning paradigm: learning using privileged information. Neural Netw. **22**(5), 544–557 (2009)
35. T.T. Wong, Parametric methods for comparing the performance of two classification algorithms evaluated by k-fold cross validation on multiple data sets. Pattern Recogn. **65**, 97–107 (2017)
36. H. Yin, ViSOM-a novel method for multivariate data projection and structure visualization. IEEE Trans. Neural Netw. **13**(1), 237–243 (2002)
37. L.A. Zadeh (ed.), Computing with words in Information/Intelligent systems 1: Foundations. Physica (2013)
38. P. Zhang, Model selection via multifold cross validation. Ann. Stat. **1**, 299–313 (1993)